Práctica 2.5. Sockets

Objetivos

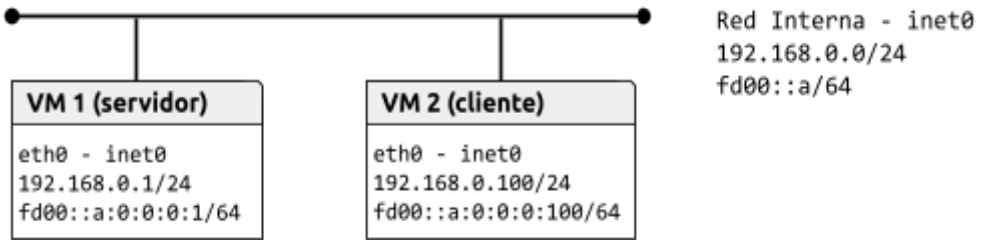
En esta práctica, nos familiarizaremos con la interfaz de programación de sockets como base para la programación de aplicaciones basadas en red, poniendo de manifiesto las diferencias de programación entre los protocolos UDP y TCP. Además, aprenderemos a programar aplicaciones independientes de la familia de protocolos de red (IPv4 o IPv6) utilizados.

Contenidos

Preparación del entorno de la práctica
Gestión de direcciones
Protocolo UDP - Servidor de hora
Protocolo TCP - Servidor de eco

Preparación del entorno de la práctica

Configuraremos la topología de red que se muestra en la figura. Como en prácticas anteriores construiremos la topología con la herramienta `vtopol`. Antes de comenzar la práctica, configurar los interfaces de red como se indica en la figura y comprobar la conectividad entre las máquinas.



Nota: Observar que las VMs tienen un interfaz de red con pila dual IPv6 - IPv4.

```
netprefix inet
machine 1 0 0
machine 2 0 0

En VM1:
ip link set eth0 up
ip a add 192.168.0.1/24 dev eth0
ip a add fd00::a:0:0:0:1/64 dev eth0
En VM2:
ip link set eth0 up
ip a add 192.168.0.100/24 dev eth0
ip a add fd00::a:0:0:0:100/64 dev eth0
```

Gestión de direcciones

El uso del API BSD requiere la manipulación de direcciones de red y la traducción de estas entre las tres representaciones básicas: nombre de dominio, dirección IP (versión 4 y 6) y binario (para incluirla en la cabecera del datagrama IP).

Ejercicio 1. Escribir un programa que obtenga todas las posibles direcciones con las que se podría crear un socket asociado a un host dado como primer argumento del programa. Para cada dirección,

mostrar la IP numérica, la familia de protocolos y tipo de socket. Comprobar el resultado para:

- Una dirección IPv4 válida (ej. "`147.96.1.9`").
- Una dirección IPv6 válida (ej. "`fd00::a:0:0:0:1`").
- Un nombre de dominio válido (ej. "`www.google.com`").
- Un nombre en `/etc/hosts` válido (ej. "`localhost`").
- Una dirección o nombre incorrectos en cualquiera de los casos anteriores.

El programa se implementará usando `getaddrinfo(3)` para obtener la lista de posibles direcciones de socket (**`struct`** `sockaddr`). Cada dirección se imprimirá en su valor numérico, usando `getnameinfo(3)` con el *flag* `NI_NUMERICHOST`, así como la familia de direcciones y el tipo de socket.

Nota: Para probar el comportamiento con DNS, realizar este ejercicio en la máquina física.

Ejemplos:

```
# Las familias 2 y 10 son AF_INET y AF_INET6, respectivamente (ver socket.h)
# Los tipos 1, 2, 3 son SOCK_STREAM, SOCK_DGRAM y SOCK_RAW, respectivamente
> ./gai www.google.com
66.102.1.147 2     1
66.102.1.147 2     2
66.102.1.147 2     3
2a00:1450:400c:c06::67    10     1
2a00:1450:400c:c06::67    10     2
2a00:1450:400c:c06::67    10     3
> ./gai localhost
::1    10     1
::1    10     2
::1    10     3
127.0.0.1    2     1
127.0.0.1    2     2
127.0.0.1    2     3
> ./gai ::1
::1    10     1
::1    10     2
::1    10     3
> ./gai 1::3::4
Error getaddrinfo(): Name or service not known
> ./gai noexiste.ucm.es
Error getaddrinfo(): Name or service not known
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <errno.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>

int main (int argc, char**argv) {
  if (argc < 2) {
```

```
    printf("error argumentos(direccion)\n");
    return -1;
  }

  struct addrinfo hints;
  struct addrinfo *result, *rp;
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_flags = AI_PASSIVE;
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = 0;
  hints.ai_protocol = 0;
  hints.ai_addr = NULL;
  hints.ai_canonname = NULL;
  hints.ai_next = NULL;
  if (getaddrinfo(argv[1], NULL, &hints, &result) != 0) {
    printf("error getaddrinfo\n");
    exit(EXIT_FAILURE);
  }
  char host[NI_MAXHOST];
  for (rp = result; rp != NULL; rp = rp->ai_next) {
    getnameinfo(rp->ai_addr,rp->ai_addrlen,host,NI_MAXHOST,NULL,0,NI_NUMERICHOST);
    printf("%s\t%i\t%i\t\n",host, rp->ai_family, rp->ai_socktype);
        }
  freeaddrinfo(result);
  return 0;
}
```

```
./ej1 localhost
::1    10    1
::1    10    2
::1    10    3
127.0.0.1     2     1
127.0.0.1     2     2
127.0.0.1     2     3
./ej1 ::1
::1    10    1
::1    10    2
::1    10    3
```

Protocolo UDP - Servidor de hora

Ejercicio 2. Escribir un servidor UDP de hora de forma que:

- La dirección y el puerto son el primer y segundo argumento del programa. Las direcciones pueden expresarse en cualquier formato (nombre de host, notación de punto…). Además, el servidor debe funcionar con direcciones IPv4 e IPv6 .
- El servidor recibirá un comando (codificado en un carácter), de forma que `'t'` devuelva la hora, `'d'` devuelve la fecha y `'q'` termina el proceso servidor.
- En cada mensaje el servidor debe imprimir el nombre y puerto del cliente, usar `getnameinfo(3)`.

Probar el funcionamiento del servidor con la herramienta Netcat (comando `nc` o `ncat`) como cliente.

Nota: Dado que el servidor puede funcionar con direcciones IPv4 e IPv6, hay que usar **`struct`** `sockaddr_storage` para acomodar cualquiera de ellas, por ejemplo, en `recvfrom(2)`.

Ejemplo:

```
$ ./time_server :: 3000
2 bytes de ::FFFF:192.168.0.100:58772
2 bytes de ::FFFF:192.168.0.100:58772
2 bytes de ::FFFF:192.168.0.100:58772
Comando X no soportado
2 bytes de ::FFFF:192.168.0.100:58772
Saliendo...
$
```

```
$ nc -u 192.168.0.1 3000
t
10:30:08 PMd
2014-01-14X
q
^C
$
```

Nota: El servidor no envía '\n', por lo que se muestra la respuesta y el siguiente comando (en negrita en el ejemplo) en la misma línea.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#include <time.h>
int main (int argc, char**argv) {
   if (argc < 3) {
      printf("error argumento dirección.\n");
      return -1;
   }
   struct addrinfo hints;
   struct addrinfo *result, *rp;
   memset(&hints, 0, sizeof(struct addrinfo));
   hints.ai_family = AF_UNSPEC;
   hints.ai_socktype = SOCK_DGRAM;
   hints.ai_flags = AI_PASSIVE;
   if (getaddrinfo(argv[1], argv[2], &hints, &result) != 0) {
      printf("error getaddrinfo.");
      exit(EXIT_FAILURE);
   }
   int sfd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
   if (bind(sfd, result->ai_addr, result->ai_addrlen) != 0) {
      printf("error bind.");
      exit(EXIT_FAILURE);
   }
   freeaddrinfo(result);
   char buf[2];
   char host[NI_MAXHOST];
   char service[NI_MAXSERV];
   struct sockaddr_storage peer_addr;
   socklen_t peer_addr_len = sizeof(peer_addr);
   while(1){
      ssize_t bytes = recvfrom(sfd, buf, 2, 0, (struct sockaddr *) &peer_addr, &peer_addr_len);
      buf[1] = '\0';
```

```
      getnameinfo((struct sockaddr *) &peer_addr, peer_addr_len, host, NI_MAXHOST, service,
NI_MAXSERV, NI_NUMERICHOST|NI_NUMERICSERV);
      printf("%i byte(s) de %s:%s\n", bytes, host, service);

      time_t tiempo = time(NULL);
      struct tm *tm = localtime(&tiempo);
      size_t max;
      char s[50];
      if (buf[0] == 't'){//hora
         size_t bytesT = strftime(s, max, "%I:%M:%S %p", tm);
                        s[bytesT] = '\0';
         sendto(sfd, s, bytesT, 0, (struct sockaddr *) &peer_addr, peer_addr_len);
      }
      else if (buf[0] =='d'){//fecha
         size_t bytesT = strftime(s, max, "%Y-%m-%d", tm);
         s[bytesT] = '\0';
         sendto(sfd, s, bytesT, 0, (struct sockaddr *) &peer_addr, peer_addr_len);
      }
      else if (buf[0] =='q'){
         printf("Saliendo...\n");
         exit(0);
      }
      else{
         printf("Comando no soportado: %d...\n", buf[0]);
      }
   }
   return 0;
}
```

En VM2: nc -u 192.168.0.1 3000 t 12:32:47 PMd 2021-12-13x q ^C	En VM1: ./a.out :: 3000 2 byte(s) de ::ffff:192.168.0.100:55072 2 byte(s) de ::ffff:192.168.0.100:55072 2 byte(s) de ::ffff:192.168.0.100:55072 Comando no soportado: 120... 2 byte(s) de ::ffff:192.168.0.100:55072 Saliendo...

Ejercicio 3. Escribir el cliente para el servidor de hora. El cliente recibirá como argumentos la dirección del servidor, el puerto del servidor y el comando. Por ejemplo, para solicitar la hora, `./time_client 192.128.0.1 3000 t`.

```
  #include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
```

```
#include <time.h>
int main (int argc, char**argv) {
   if (argc < 4) {
      printf("error argumentos dirección.\n");
      return -1;
   }
   struct addrinfo hints;
   struct addrinfo *result, *rp;
   memset(&hints, 0, sizeof(struct addrinfo));
   hints.ai_family = AF_UNSPEC;
   hints.ai_socktype = SOCK_DGRAM;
   hints.ai_flags = AI_PASSIVE;
   if (getaddrinfo(argv[1], argv[2], &hints, &result) != 0) {
      printf("error getaddrinfo.");
      exit(EXIT_FAILURE);
   }
   int sfd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
   freeaddrinfo(result);
   char buf[2];
   char host[NI_MAXHOST];
   char serv[NI_MAXSERV];
   struct sockaddr_storage peer_addr;
   socklen_t peer_addr_len = sizeof(peer_addr);
   sendto(sfd, argv[3], 2, 0, result->ai_addr, result->ai_addrlen);
   printf("%s\n", argv[3]);
   if (*argv[3] == 'd' || *argv[3] == 't'){
      char s[256];
      ssize_t bytes = recvfrom(sfd, s, 256, 0, (struct sockaddr *) &peer_addr, &peer_addr_len);
      s[bytes] = '\0';
      printf("ejecutaste comando %s\n", s);
   }
}
```

En Vm1:	En VM2:
./a.out :: 3000 2 byte(s) de ::ffff:192.168.0.100:53162 2 byte(s) de ::ffff:192.168.0.100:46349 2 byte(s) de ::ffff:192.168.0.100:44191 Saliendo...	./a.out 192.168.0.1 3000 t t ejecutaste comando 12:03:29 PM ./a.out 192.168.0.1 3000 d d ejecutaste comando 2021-12-15 ./a.out 192.168.0.1 3000 q q

Ejercicio 4. Modificar el servidor para que, además de poder recibir comandos por red, los pueda recibir directamente por el terminal, leyendo dos caracteres (el comando y '\n') de la entrada estándar. Multiplexar el uso de ambos canales usando `select(2)`.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#include <time.h>
int main (int argc, char**argv) {
    if (argc < 3) {
        printf("error argumento dirección.\n");
        return -1;
    }
    struct addrinfo hints;
    struct addrinfo *result, *rp;
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_flags = AI_PASSIVE;
    if (getaddrinfo(argv[1], argv[2], &hints, &result) != 0) {
        printf("error getaddrinfo.");
        exit(EXIT_FAILURE);
    }
   int sfd = socket(result->ai_family, result->ai_socktype,
result->ai_protocol);
    if (bind(sfd, result->ai_addr, result->ai_addrlen) != 0) {
        printf("error bind.");
        exit(EXIT_FAILURE);
    }
    freeaddrinfo(result);
    char buf[2];
    char host[NI_MAXHOST];
    char service[NI_MAXSERV];
    struct sockaddr_storage peer_addr;
    socklen_t peer_addr_len = sizeof(peer_addr);
    fd_set lect;
    int l =-1;
    while(1){
        while(l == -1) {
            FD_ZERO(&lect);
            FD_SET(sfd, &lect);
            FD_SET(0, &lect);
            l = select(sfd+1, &lect, NULL, NULL, NULL);
        }
        time_t tiempo = time(NULL);
        struct tm *tm = localtime(&tiempo);
        size_t max= 50;
        char s[50];

        if (FD_ISSET(sfd,&lect)){
            ssize_t bytes = recvfrom(sfd, buf, 2, 0, (struct sockaddr
*) &peer_addr, &peer_addr_len);
            buf[1] = '\0';
            getnameinfo((struct sockaddr *) &peer_addr,
```

```
peer_addr_len, host, NI_MAXHOST, service, NI_MAXSERV,
NI_NUMERICHOST|NI_NUMERICSERV);
            printf("Red %i byte(s) de %s:%s\n", bytes, host,
service);
            if (buf[0] == 't'){//hora
                size_t bytesT = strftime(s, max, "%I:%M:%S %p", tm);
                s[bytesT] = '\0';
                sendto(sfd, s, bytesT, 0, (struct sockaddr *)
&peer_addr, peer_addr_len);
            }
            else if (buf[0] =='d'){//fecha
                size_t bytesT = strftime(s, max, "%Y-%m-%d", tm);
                s[bytesT] = '\0';
                sendto(sfd, s, bytesT, 0, (struct sockaddr *)
&peer_addr, peer_addr_len);
            }
            else if (buf[0] =='q'){
                printf("Saliendo...\n");
                exit(0);
            }
            else{
                printf("Comando no soportado: %d...\n", buf[0]);
            }
        }
        else{
            read(0,buf,2);
            buf[1] = '\0';
            printf("Consola %i byte(s)\n", 1);
            if (buf[0] == 't'){
                size_t bytesT = strftime(s, max, "%I:%M:%S %p", tm);
                s[bytesT] = '\0';
                printf("%s\n",s);
            }
            else if (buf[0] =='d'){
                size_t bytesT = strftime(s, max, "%Y-%m-%d", tm);
                s[bytesT] = '\0';
                printf("%s\n",s);
            }
            else if (buf[0] =='q'){
                printf("Saliendo...\n");
                exit(0);
            }else{
                printf("Comando no soportado: %d...\n", buf[0]);
            }
        }
    }
    close(sfd);
    return 0;
}
```

En VM1: ./a.out :: 3000	En VM2: nc -u 192.168.0.1 3000

REDES: 2 byte(s) de ::ffff:192.168.0.100:59896 REDES: 2 byte(s) de ::ffff:192.168.0.100:59896 t CONSOLA: 1 byte(s) 06:15:48 PM d CONSOLA: 1 byte(s) 2021-12-16 q CONSOLA: 1 byte(s) Saliendo...	t 06:15:25 PMd 2021-12-16q Ncat: Connection refused.(2º q)

Ejercicio 5. Convertir el servidor UDP en multi-proceso siguiendo el patrón *pre-fork*. Una vez asociado el socket a la dirección local con `bind(2)`, crear varios procesos que llamen a `recvfrom(2)` de forma que cada uno atenderá un mensaje de forma concurrente. Imprimir el PID del proceso servidor para comprobarlo. Para terminar el servidor, enviar la señal `SIGTERM` al grupo de procesos.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#include <time.h>
#include <signal.h>
int main (int argc, char**argv) {
   if (argc < 3) {
      printf("error argumentos direccion\n");
      return -1;
   }
   struct addrinfo hints;
   struct addrinfo *result, *rp;
   memset(&hints, 0, sizeof(struct addrinfo));
   hints.ai_family = AF_UNSPEC;
   hints.ai_socktype = SOCK_DGRAM;
   hints.ai_flags = AI_PASSIVE;
   if (getaddrinfo(argv[1], argv[2], &hints, &result) != 0) {
      printf("error getaddrinfo.");
      exit(EXIT_FAILURE);
   }
   int sfd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
   if (bind(sfd, result->ai_addr, result->ai_addrlen) != 0) {
      printf("error bind.");
      exit(EXIT_FAILURE);
   }
   freeaddrinfo(result);
   char buf[2];
```

```
   char host[NI_MAXHOST];
   char serv[NI_MAXSERV];
   struct sockaddr_storage peer_addr;
   socklen_t peer_addrlen = sizeof(peer_addr);
   int i = 0;
   while(1){
      for (i = 0; i < 3; i++){
         pid_t pid;
         pid = fork();
         if(pid == -1){
            perror("error fork\n");
            exit(1);
         }
         if (pid == 0) {
            ssize_t bytes = recvfrom(sfd, buf, 2, 0, (struct sockaddr *) &peer_addr, &peer_addrlen);
            buf[1] = '\0';
            getnameinfo((struct sockaddr *) &peer_addr, peer_addrlen, host, NI_MAXHOST, serv,
NI_MAXSERV, NI_NUMERICHOST|NI_NUMERICSERV);
            printf("PID: %d recibe %i byte(s) de %s:%s\n", getpid(), bytes, host, serv);
            time_t tiempo = time(NULL);
            struct tm *tm = localtime(&tiempo);
            size_t max=50;
            char s[50];
            if (buf[0] == 't'){
               size_t bytesT = strftime(s, max, "%I:%M:%S %p", tm);
               s[bytesT] = '\0';
               sendto(sfd, s, bytesT, 0, (struct sockaddr *) &peer_addr, peer_addrlen);
            }
            else if (buf[0] =='d'){
               size_t bytesT = strftime(s, max, "%Y-%m-%d", tm);
               s[bytesT] = '\0';
               sendto(sfd, s, bytesT, 0, (struct sockaddr *) &peer_addr, peer_addrlen);
            }
            else if (buf[0] =='q'){
               printf("Saliendo...\n");
               kill(0,SIGTERM);
            }
            else{
               printf("Comando no soportado: %d...\n", buf[0]);
            }
         }
         else {
            wait(NULL);
         }
      }
   }
   close (sfd);
   return 0;
}
```

En VM1: ./a.out :: 3000 PID: 3512 recibe 2 byte(s) de ::ffff:192.168.0.100:38346 PID: 3515 recibe 2 byte(s) de ::ffff:192.168.0.100:44014	nc -u 192.168.0.1 3000 t 07:42:13 PM ^C	nc -u192.168.0.1 3000 d 2021-12-16q ^C

PID: 3516 recibe 2 byte(s) de ::ffff:192.168.0.100:44014 Saliendo... Terminated		

Protocolo TCP - Servidor de eco

TCP nos ofrece un servicio orientado a conexión y fiable. Una vez creado el socket, debe ponerse en estado LISTEN (apertura pasiva, `listen(2)`) y a continuación quedarse a la espera de conexiones entrantes mediante una llamada `accept(2)`.

Ejercicio 6. Crear un servidor TCP de eco que escuche por conexiones entrantes en una dirección (IPv4 o IPv6) y puerto dados. Cuando reciba una conexión entrante, debe mostrar la dirección y número de puerto del cliente. A partir de ese momento, enviará al cliente todo lo que reciba desde el mismo (eco). Comprobar su funcionamiento empleando la herramienta Netcat como cliente. Comprobar qué sucede si varios clientes intentan conectar al mismo tiempo.

Ejemplo:

`$ ./echo_server :: 2222`
`Conexión desde fd00::a:0:0:0:100 53456`
`Conexión terminada`
	`$ nc -6 fd00::a:0:0:0:1 2222`
	`Hola`
	`Hola`
	`Qué tal`
	`Qué tal`
	`^C`
	`$`

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int main(int argc, char**argv){
   if (argc < 3) {
      printf("error parámetros\n");
      return -1;
   }
   struct addrinfo hints;
   struct addrinfo *result;
   memset(&hints,0,sizeof(struct addrinfo));
   hints.ai_family = AF_UNSPEC;
   hints.ai_socktype = SOCK_STREAM;
   hints.ai_flags = AI_PASSIVE;
   if (getaddrinfo(argv[1], argv[2], &hints, &result) != 0) {
      printf("error getaddrinfo");
      exit(EXIT_FAILURE);
   }
   int sfd = socket(result->ai_family, result->ai_socktype, 0);
   if(bind(sfd, result->ai_addr, result->ai_addrlen)!=0){
      printf("error bind");
      exit(EXIT_FAILURE);
   }
   freeaddrinfo(result);
```

```
   if(listen(sfd, 5)==-1){
      printf("error listen");
      exit(1);
   }
   struct sockaddr_storage peer_addr;
   socklen_t peer_addr_len = sizeof(peer_addr);
   int peer_addr_sd;
   char host[NI_MAXHOST];
   char serv[NI_MAXSERV];
   char buf[256];
   size_t max = 256;
   while (1) {
      peer_addr_sd = accept(sfd,(struct sockaddr *) &peer_addr, &peer_addr_len);
      ssize_t bytesT;
      getnameinfo((struct sockaddr *)&peer_addr, peer_addr_len, host, NI_MAXHOST, serv,
NI_MAXSERV, NI_NUMERICHOST);
      while (bytesT= recv(peer_addr_sd, buf, max, 0)) {
         buf[bytesT] = '\0';
         printf("Conexión establecida con %s:%s\n", host, serv);
         send(peer_addr_sd, buf, bytesT, 0);
      }
      printf("Conexion terminada\n");
      close(peer_addr_sd);
   }
   return 0;
}
```

Solo permite interactuar al primer qliente en conectar

Servidor:	Cliente1:	Cliente 2:
./a.out :: 2000 Conexión establecida con fd00:0:0:a::100:45162 Conexión establecida con fd00:0:0:a::100:45162 Conexion terminada Conexión establecida con fd00:0:0:a::100:45164 Conexión establecida con fd00:0:0:a::100:45164 Conexión establecida con fd00:0:0:a::100:45164 Conexion terminada	nc -6 fd00::a:0:0:0:1 2000 hola soy C1 y llego primero hola soy C1 y llego primero no dejo a c2 no dejo a c2 ^C	nc-6 fd00::a:0:0:0:1 2000 no puedo hasta que se vaya C1 no puedo hasta que se vaya C1 ya puedo ya puedo adios adios ^C

Ejercicio 7. Escribir el cliente para conectarse con el servidor del ejercicio anterior. El cliente recibirá la dirección y el puerto del servidor como argumentos y, una vez establecida la conexión con el servidor, le enviará lo que el usuario escriba por teclado. Mostrará en la consola la respuesta recibida desde el servidor. Cuando el usuario escriba el carácter 'Q' como único carácter de una línea, el cliente cerrará la conexión con el servidor y terminará.

Ejemplo:

`$ ./echo_server :: 2222` `Conexión desde fd00::a:0:0:0:100 53445`	`$ ./echo_client fd00::a:0:0:0:1 2222` **`Hola`**

`Conexión terminada`	`Hola`
	`Q`
	`$`

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int main(int argc, char**argv){
   if (argc < 3) {
      printf("error parámetros.\n");
      return -1;
   }
   struct addrinfo hints;
   struct addrinfo *result;
   memset(&hints,0,sizeof(struct addrinfo));
   hints.ai_family = AF_UNSPEC;
   hints.ai_socktype = SOCK_STREAM;
   hints.ai_flags = AI_PASSIVE;
   if (getaddrinfo(argv[1], argv[2], &hints, &result) != 0) {
      printf("error getaddrinfo\n");
      exit(EXIT_FAILURE);
   }
   int sfd = socket(result->ai_family, result->ai_socktype, 0);
   if(connect(sfd,(struct sockaddr *)result->ai_addr, result->ai_addrlen)==-1){
      printf("error connect\n");
      exit(EXIT_FAILURE);
   }
   freeaddrinfo(result);
   char buf[256];
   size_t max = 256;
   ssize_t bytesT;
   while (1) {
      bytesT = read(0,buf, 255);
      buf[bytesT] = '\0';
      send(sfd,buf, bytesT, 0);
      if ((buf[0] == 'Q') && (bytesT == 2)) {
         printf("Conexión terminada\n");
         break;
      }
      bytesT = recv(sfd, buf, bytesT, 0);
      buf[bytesT] = '\0';
      printf("%s\n", buf);
   }
   close(sfd);
   return 0;
}
```

En Vm1: serv ./ej6 :: 2222 Conexión establecida con fd00:0:0:a::100:44448	En VM2: cliente (ej7) ./ej7 fd00::a:0:0:0:1 2222 Hola soy el cliente primero

Conexión establecida con fd00:0:0:a::100:44448 Conexión establecida con fd00:0:0:a::100:44448 Conexion terminada ^C	Hola soy el cliente primero Un placer conocerte Un placer conocerte Q Conexión terminada

Ejercicio 8. Modificar el código del servidor para que acepte varias conexiones simultáneas. Cada petición debe gestionarse en un proceso diferente, siguiendo el patrón *accept-and-fork*. El proceso padre debe cerrar el socket devuelto por `accept(2)`.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int main(int argc, char**argv){
   if (argc < 3) {
      printf("error parámetros.\n");
      return -1;
   }
   struct addrinfo hints;
   struct addrinfo *result;
   memset(&hints,0,sizeof(struct addrinfo));
   hints.ai_family = AF_UNSPEC;
   hints.ai_socktype = SOCK_STREAM;
   hints.ai_flags = AI_PASSIVE;
   if (getaddrinfo(argv[1], argv[2], &hints, &result) != 0) {
      printf("error getaddrinfo\n");
      exit(EXIT_FAILURE);
   }
   int sfd = socket(result->ai_family, result->ai_socktype, 0);
   if(connect(sfd,(struct sockaddr *)result->ai_addr, result->ai_addrlen)==-1){
      printf("error connect\n");
      exit(EXIT_FAILURE);
   }
   freeaddrinfo(result);
   char buf[256];
   size_t max = 256;
   ssize_t bytesT;
   while (1) {
      bytesT = read(0,buf, 255);
      buf[bytesT] = '\0';
      send(sfd,buf, bytesT, 0);
      if ((buf[0] == 'Q') && (bytesT == 2)) {
         printf("Conexión terminada\n");
         break;
      }
      bytesT = recv(sfd, buf, bytesT, 0);
      buf[bytesT] = '\0';
```

```
    printf("%s\n", buf);
  }
  close(sfd);
  return 0;
}
```

SERV en VM1:	C1 en VM2:
./ej8 :: 2222 PID: 3950 conectado con fd00:0:0:a::100:55466 PID: 3959 conectado con fd00:0:0:a::100:55468 Conexión terminada PID: 3970 conectado con fd00:0:0:a::100:55470 Conexión terminada Conexión terminada ^C	./ej7 fd00::a:0:0:0:1 2222 Hola llego primero ahora viene C2 Hola llego primero ahora viene C2 Tengo que irme Tengo que irme Q Conexión terminada
C2 en VM2:	**C3 en VM2:**
./ej7 fd00::a:0:0:0:1 2222 Ya llegue Ya llegue Yo me marcho cuando llegue C3 Yo me marcho cuando llegue C3 Q Conexión terminada	./ej7 fd00::a:0:0:0:1 2222 Te puedes ir ya C2 Te puedes ir ya C2 Me despido Me despido Q Conexión terminada

Ejercicio 9. Añadir la lógica necesaria en el servidor para que no quede ningún proceso en estado *zombie*. Para ello, se deberá capturar la señal `SIGCHLD` y obtener la información de estado de los procesos hijos finalizados.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

#include <signal.h>
void handler(int signal){
  int status;
  wait(&status);
  printf("Proceso finalizado: %i\n", status);
}
int main(int argc, char**argv){
  if (argc < 3) {
    printf("error parámetros\n");
    return -1;
  }
```

```
   struct addrinfo hints;
   struct addrinfo *result;
   memset(&hints,0,sizeof(struct addrinfo));
   hints.ai_family = AF_UNSPEC;
   hints.ai_socktype = SOCK_STREAM;
   hints.ai_flags = AI_PASSIVE;
   if (getaddrinfo(argv[1], argv[2], &hints, &result) != 0) {
      printf("error getaddrinfo");
      exit(EXIT_FAILURE);
   }
   int sfd = socket(result->ai_family, result->ai_socktype, 0);
   if(bind(sfd, result->ai_addr, result->ai_addrlen)!=0){
      printf("error bind");
      exit(EXIT_FAILURE);
   }
   freeaddrinfo(result);
   if(listen(sfd, 5)==-1){
      printf("error listen");
      exit(1);
   }
   struct sockaddr_storage peer_addr;
   socklen_t peer_addr_len = sizeof(peer_addr);
   int peer_addr_sd;
   char host[NI_MAXHOST];
   char serv[NI_MAXSERV];
   char buf[256];
   size_t max = 256;
   struct sigaction act;
   act.sa_handler= handler;
   while (1) {
      peer_addr_sd = accept(sfd,(struct sockaddr *) &peer_addr, &peer_addr_len);
      ssize_t bytes;
      pid_t pid = fork();
      if (pid == -1){
         printf("error fork\n");
         exit(EXIT_FAILURE);
      }
      else if (pid == 0){
         getnameinfo((struct sockaddr *)&peer_addr, peer_addr_len, host, NI_MAXHOST, serv,
NI_MAXSERV, NI_NUMERICHOST);
         printf("PID: %d conectado con %s:%s\n", getpid(), host, serv);
         while(bytes= recv(peer_addr_sd, buf, max, 0)){
            buf[bytes] = '\0';
            if ((buf[0] == 'Q') && (bytes == 2)) {
               printf("Conexión terminada\n");
               exit(0);
            }
            printf("PID: %d recibe %i byte(s) de %s:%s\n", getpid(), bytes, host, serv);
            send(peer_addr_sd, buf, bytes, 0);
         }
         kill(getppid(),SIGCHLD);
      }
      else {
         sigaction(SIGCHLD,&act,NULL);
         close(peer_addr_sd);
      }
   }
```

```
    return 0;
}
```

Serv en VM1:
./a.out :: 2222
PID: 4477 conectado con fd00:0:0:a::100:44464
PID: 4477 recibe 20 byte(s) de fd00:0:0:a::100:44464
PID: 4478 conectado con fd00:0:0:a::100:44466
PID: 4478 recibe 17 byte(s) de fd00:0:0:a::100:44466
PID: 4477 recibe 24 byte(s) de fd00:0:0:a::100:44464
Conexión terminada
Proceso finalizado: 0
PID: 4478 recibe 12 byte(s) de fd00:0:0:a::100:44466
Conexión terminada
Proceso finalizado: 0

^C

C1 en VM2:	C2 en VM2
./a.out fd00::a:0:0:0:1 2222	./a.out fd00::a:0:0:0:1 2222
Hola ahora viene C2	Saludos mi gente
Hola ahora viene C2	Saludos mi gente
Me marcho y me sigue C2	Arrivederci
Me marcho y me sigue C2	Arrivederci
Q	Q
Conexión terminada	Conexión terminada